TAPNIO, SHERWIN CLYDE
December 4. 2025 C203

Finals Task 7. Tkinter CRUD GUI

main.py

```python
import tkinter as tk
import window

def main():
    root = tk.Tk()
    # Passing the root to the Window class
    crud = window.Window(root)
    root.mainloop()

if __name__ == "__main__":
    main()
```

connectDB.py

```python
import mysql.connector
from tkinter import messagebox

class ConnectDB:
    def __init__(self, host, user, password, database):
        self.host = host
        self.user = user
        self.password = password
        self.database = database
        self.connection = None  # Renamed to avoid confusion with method name

    def connect(self):
        try:
            self.connection = mysql.connector.connect(
                host="localhost",
                user="root",
                password="",
                database="cars", # Ensure your MySQL database is named 'cars'
                ssl_disabled=True
            )
            print("Successfully connected to the database!")
        except mysql.connector.Error as error:
            print("Something went wrong connecting to the database: ", error)

    def disconnect(self):
        if self.connection:
            self.connection.close()
            print("Successfully disconnected from the database!")

    def execute_insert(self, table, id, model, year, color, capacity, power, typ
        sql = f"INSERT INTO {table} (id, model, year, color, engineCapacity, eng
        self.commit_to_db(sql)
```

```python
    def execute_delete(self, table, id):
        sql = f"DELETE FROM {table} WHERE id = {id}"
        self.commit_to_db(sql)

    def execute_update(self, table, id, model, year, color, capacity, power, engi
        sql = f"UPDATE {table} SET model='{model}', year='{year}', color='{color}
        self.commit_to_db(sql)

    def commit_to_db(self, sql):
        try:
            cursor = self.connection.cursor()
            cursor.execute(sql)
            self.connection.commit()
            print("Query successfully executed")
            messagebox.showinfo("Success", "Query successfully executed. Good Wor
        except mysql.connector.Error as error:
            self.connection.rollback()
            print("Error executing the query:", error)
            messagebox.showerror("Error", f"Error executing query: {error}")

    def execute_select(self, table):
        sql = f"SELECT * FROM {table}"
        try:
            cursor = self.connection.cursor()
            cursor.execute(sql)
            rows = cursor.fetchall()
            return rows
        except mysql.connector.Error as error:
            print("Error executing the query:", error)
            return []
```

window.py

```python
import tkinter as tk
from tkinter import font
from tkinter import ttk
from connectDB import *
from tkinter import messagebox

class Window:
    def __init__(self, root):
        self.root = root
        # Initialize DB connection object
        self.cnn = ConnectDB(host="localhost", user="root", password="", databas
        self.settings()
        self.create_widgets()

    def settings(self):
        # UPDATED NAME HERE
        self.root.title("CRUD PYTHON MYSQL - BMWCars - TAPNIO, SHERWIN CLYDE")
        self.root.resizable(0, 0)

        # Center the window
        widthScreen = self.root.winfo_screenwidth()
        heightScreen = self.root.winfo_screenheight()
        widthWindow = 1200
        heightWindow = 600
        pwidth = int(widthScreen / 2 - widthWindow / 2)
        pheight = int(heightScreen / 2 - heightWindow / 2)
        self.root.geometry(f"{widthWindow}x{heightWindow}+{pwidth}+{pheight}")

    def create_widgets(self):
        # --- FRAME BUTTONS ---
        frame1 = tk.Frame(self.root, width=200, height=600, bg="#f7f5f0")
        frame1.place(x=0, y=0)

        self.buttonInit = tk.Button(frame1, text="Show All", command=self.fnInit
        self.buttonInit.place(x=10, y=20)
```

```
        self.buttonNew = tk.Button(frame1, text="Add Record", command=self.Insert
        self.buttonNew.place(x=10, y=100)

        self.buttonUpdate = tk.Button(frame1, text="Update", command=self.Update[
        self.buttonUpdate.place(x=10, y=150)

        self.buttonDelete = tk.Button(frame1, text="Delete", command=self.Delete[
        self.buttonDelete.place(x=10, y=200)

        self.buttonSearch = tk.Button(frame1, text="Search", command=self.Search[
        self.buttonSearch.place(x=10, y=250)

        self.buttonReload = tk.Button(frame1, text="Reload", command=self.fnInit,
        self.buttonReload.place(x=10, y=300)

        # Stats Label
        self.stats_label = tk.Label(frame1, text="Stats will appear here", bg="#
        self.stats_label.place(x=10, y=360)

        # --- FRAME INPUT ---
        self.frame2 = tk.Frame(self.root, width=300, height=600, bg="#CCCCCC")

        # Labels and Entries
        labels = ["ID", "Model:", "Year Make:", "Color:", "Engine Capacity:", "Er
        self.entries = []

        y_pos = 15
        for i, label_text in enumerate(labels):
            lbl = tk.Label(self.frame2, text=label_text, background="#CCCCCC")
            lbl.place(x=10, y=y_pos)
            entry = tk.Entry(self.frame2, width=30, font=font.Font(size=12))
            entry.place(x=10, y=y_pos + 25)
            self.entries.append(entry)
            y_pos += 65
```

```
            entry.place(x=10, y=y_pos + 25)
            self.entries.append(entry)
            y_pos += 65

        # Map entries to self variables for easier access later
        self.entry1, self.entry2, self.entry3, self.entry4, self.entry5, self.ent

        self.buttonSave = tk.Button(frame1, text="Save", command=self.save, width
        self.buttonCancel = tk.Button(frame1, text="Cancel", command=self.cancel,

        # --- TREEVIEW GRID ---
        style = ttk.Style()
        style.configure("Custom.Treeview", background="whitesmoke", foreground="b

        columns = ("col1", "col2", "col3", "col4", "col5", "col6", "col7", "col8"
        self.grid = ttk.Treeview(self.root, columns=columns, style="Custom.Treevi

        self.grid.column("#0", width=50, anchor=tk.CENTER)
        self.grid.heading("#0", text="ID")

        headers = ["Model", "Year", "Color", "Capacity", "Power", "Type", "Trans"
        for col, header in zip(columns, headers):
            self.grid.column(col, width=90, anchor=tk.CENTER)
            self.grid.heading(col, text=header)

        self.grid.place(x=200, y=0, width=999, height=599)

    def update_stats(self):
        self.cnn.connect()
        data = self.cnn.execute_select("car")
        self.cnn.disconnect()

        if not data:
            self.stats_label.config(text="No records found.")
            return

        total_records = len(data)
```

```
        # Calculate highest price safely
        try:
            highest_price_row = max(data, key=lambda x: x[8])
            highest_price_model = highest_price_row[1]
        except:
            highest_price_model = "N/A"

        total_manual = sum(1 for row in data if str(row[7]).lower() == "manual"
        total_auto = sum(1 for row in data if str(row[7]).lower() == "automatic"

        stats_text = (
            f"Total Records: {total_records}\n"
            f"Highest Price Model: {highest_price_model}\n"
            f"Total Manual: {total_manual}\n"
            f"Total Automatic: {total_auto}"
        )
        self.stats_label.config(text=stats_text)

    def fnInit(self):
        self.grid.delete(*self.grid.get_children())
        self.cnn.connect()
        data = self.cnn.execute_select("car")
        self.cnn.disconnect()

        for row in data:
            self.grid.insert("", tk.END, text=row[0], values=row[1:])

        self.buttonInit.config(state="disabled")
        self.update_stats()

    def cancel(self):
        self.buttonSave.place_forget()
        self.buttonCancel.place_forget()
        self.grid.place(x=200, y=0, width=999, height=599)
        self.frame2.place_forget()
```

```
        # Clear entries
        for entry in self.entries:
            entry.config(state="normal")
            entry.delete(0, tk.END)

        self.buttonUpdate.config(state="normal")
        self.buttonNew.config(state="normal")
        self.buttonDelete.config(state="normal")
        self.buttonSearch.config(state="normal")
        self.buttonReload.config(state="normal")

    def save(self):
        try:
            val_id = int(self.entry1.get())
            val_model = self.entry2.get()
            val_year = self.entry3.get()
            val_color = self.entry4.get()
            val_capacity = int(self.entry5.get())
            val_power = int(self.entry6.get())
            val_type = self.entry7.get()
            val_trans = self.entry8.get()
            val_price = float(self.entry9.get())

            if not val_model or not val_type:
                messagebox.showerror("Error", "All fields must be filled in.")
                return

            self.cnn.connect()

            if self.entry1.cget("state") == "normal": # Insert
                self.cnn.execute_insert("car", val_id, val_model, val_year, val_
            else: # Update
                self.cnn.execute_update("car", val_id, val_model, val_year, val_

            self.cnn.disconnect()
            self.cancel() # Reset view
            self.fnInit() # Reload data
```

```
            self.cnn.execute_update("car", val_id, val_model, val_year, val_(

            self.cnn.disconnect()
            self.cancel() # Reset view
            self.fnInit() # Reload data

        except ValueError:
            messagebox.showerror("Error", "Check your inputs. ID, Capacity, Powe

    def InsertData(self):
        self.grid.place(x=500, y=0, width=699, height=599)
        self.frame2.place(x=200, y=0)
        self.buttonSave.place(x=10, y=495) # Adjusted position
        self.buttonCancel.place(x=10, y=545)

        self.buttonUpdate.config(state="disabled")
        self.buttonNew.config(state="disabled")
        self.buttonDelete.config(state="disabled")
        self.buttonSearch.config(state="disabled")
        self.buttonReload.config(state="disabled")

    def UpdateData(self):
        selection = self.grid.selection()
        if selection:
            self.InsertData() # Setup layout like Insert

            id_selected = self.grid.item(selection)['text']
            values = self.grid.item(selection)['values']

            self.entry1.insert(0, id_selected)
            self.entry1.config(state="disabled") # ID cannot be changed on update

            # Fill other fields
            for i, val in enumerate(values):
                self.entries[i+1].insert(0, val)
        else:
```

```
        messagebox.showerror("Error", "You must select a record to update")

    def DeleteData(self):
        selection = self.grid.selection()
        if selection:
            id_selected = self.grid.item(selection)['text']
            confirm = messagebox.askyesno("Confirm", "Are you sure you want to de
            if confirm:
                self.cnn.connect()
                self.cnn.execute_delete("car", id_selected)
                self.cnn.disconnect()
                self.fnInit()
        else:
            messagebox.showerror("Error", "You must select a record to delete")

    def SearchData(self):
        new_window = tk.Toplevel(self.root)
        new_window.title("Search")
        new_window.geometry("600x100")

        radio_var = tk.StringVar(value="option1")

        ttk.Radiobutton(new_window, text="Id", variable=radio_var, value="option1
        ttk.Radiobutton(new_window, text="Model", variable=radio_var, value="opti
        ttk.Radiobutton(new_window, text="Year", variable=radio_var, value="optio
        ttk.Radiobutton(new_window, text="Price", variable=radio_var, value="opti

        entry_search = tk.Entry(new_window, width=30)
        entry_search.place(x=320, y=14)

        def run_search():
            search_text = entry_search.get().lower()
            option = radio_var.get()

            index_map = {"option1": 0, "option2": 1, "option3": 2, "option4": 8}
            col_index = index_map.get(option, 0)
```

```
        option = radio_var.get()

        index_map = {"option1": 0, "option2": 1, "option3": 2, "option4": 8
        col_index = index_map.get(option, 0)

        self.cnn.connect()
        data = self.cnn.execute_select("car")
        self.cnn.disconnect()

        found_items = []
        for row in data:
            # Check ID (index 0) separately as it might be int
            check_val = str(row[col_index]).lower()
            if search_text in check_val:
                found_items.append(row)

        # Update grid
        self.grid.delete(*self.grid.get_children())
        for row in found_items:
            self.grid.insert("", tk.END, text=row[0], values=row[1:])

        new_window.destroy()

    btn = ttk.Button(new_window, text="Search", command=run_search)
    btn.place(x=520, y=11)
```